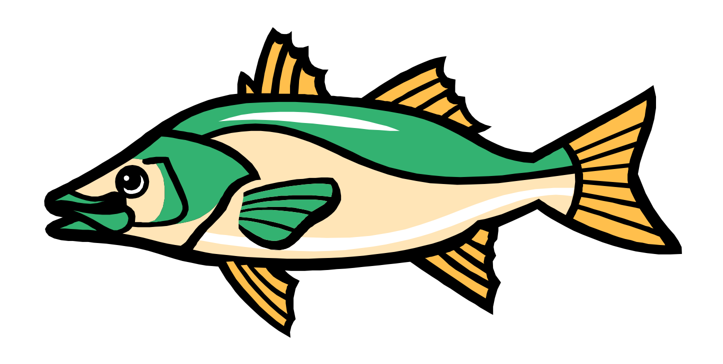
Einführung in die Programmierung mit

mscFish

VBScript Version

Ulrich Müller



Inhaltsverzeichnis

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873, Fax 05251/55709
eMail : UM@ftcomputing.de
HomePage : www.ftcomputing.de

Dokumentname : mscFishVBS.doc. Druckdatum : 07.08.2003
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish12.WMF

Übersicht

Allgemeines

mscFish ist eine IDE (Integrated Development Environment) für ftComputing auf Basis der Microsoft Sprachen VBScript, JScript(JavaScript) und des ActiveX FishFa30.DLL. Dabei wurden die Sprachelemente von FishFa30.DLL nahtlos in die Sprache VBScript JScript integriert. In diesem Hanbuch wird der Umgang mit VBScript beschrieben. Zu JScript siehe mscFish30JS.PDF.

mscFish wurde bewußt einfach gestaltet um Programmier-Anfängern ein einfaches und, in allen seinen Komponenten, *kostenloses* Werkzeug zur Verfügung zu stellen, mit dem im Selbststudium der Einstieg in die Welt der Programmierens gefunden werden kann.

Deshalb ist der Abschnitt "Einführung in die Programmierung" der wesentliche Teil dieses Buchs. Man arbeitet ihn am besten in der vorgegebenen Reihenfolge durch. Parallel dazu sollte man sich mit der Microsoft Dokumentation zu VBScript (siehe "Hilfe – Dokumentation") vertraut machen.

mscFish ist geeignet für Programme im Umfang von ein paar Zeilen bis zu ein paar Seiten. Die Möglichkeiten der Kommunikation mit dem Bediener sind beit VBScript beschränkt, aber in diesem Rahmen in Verbindung mit der IDE voll ausreichend. Ist man aus diesem Rahmen herausgewachsen, sollte man auf "größere" Sprachen der Visual Basic Sprachfamilie umsteigen. Das ist problemlos möglich. Siehe Anhang.

Installation

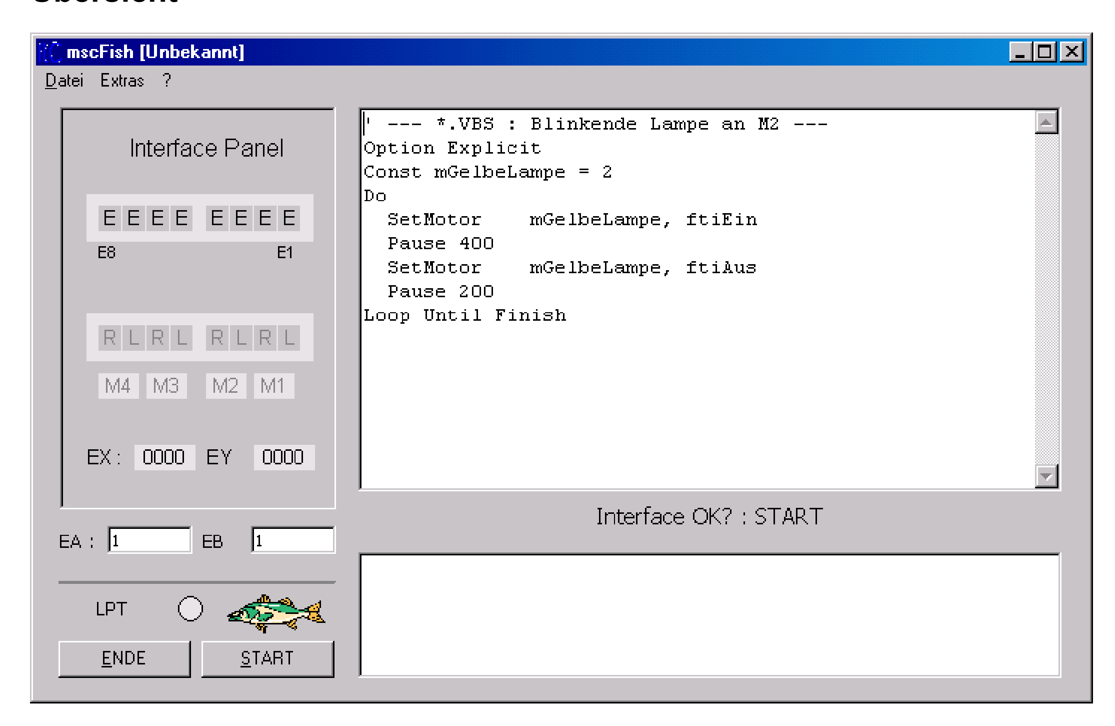
mscFish wird in Form eines Setup-Programmes : mscFish30Setup.EXE ausgeliefert. Die Installation erfolgt weitgehend automatisch nach den üblichen Regeln.

mscFish läuft ab Windows 98 und setzt ein installiertes Visual Basic 6 Laufzeitsystem (MSVBVM60.DLL) und einen installierten und aktivierten WSH (Windows Scripting Host einschließlich VBScript-Runtime) voraus. Siehe auch Anhang "Übersicht mscFish".

Wenn ein altes Universal-Interface am LPT-Port betrieben werden soll, ist bei der Installation EINE der LPT-Optionen auszuwählen.

IDE – Integrated Development Environment

Übersicht



Die IDE gliedert sich in folgende Teile :

1. Bedienung der Gesamt IDE : Buttons links unter dem Strich
2. Kommunikation mit dem Modell über das Interface : Interface Panel
3. Parameter für die Anwendung : Die Felder EA und EB
4. Anzeige und Editieren der Anwendung : Editier-Feld, rechts oben
5. Protokollierung des Programmablaufs : Log-Feld, rechts unten
6. Anzeige des Programmstatus : Die Statuszeile, dazwischen
7. Laden und Speichern von Anwendungen : Menü Datei
8. Dokumentation : Script56.CHM für VBScript und mscFish.PDF für FishFace z.Zt. als separate Dateien.
9. Fehleranzeigen : über Message Boxen

Bedienung der IDE

1. Einstellen der Interface-Verbindung über das Menü Extras | Interface Optionen. Der eingestellte Wert wird links unten über dem ENDE-Button angezeigt und bei Programmende.
2. START-Button klicken.
3. Eine erfolgreiche Interface-Verbindung wird über einen grünen Button links neben der Interface-Verbindung angezeigt, die Beschriftung des START-Buttons wechselt in RUN, in der Statuszeile steht Editier-Modus. Anderfalls eine MessageBox : "Fehler beim Öffnen des Interfaces". Oft ist dann das Netzteil nicht angeschlossen.
4. Bei erfolgreicher Interface-Verbindung kann das Interface Panel genutzt werden. Getätigte E-Eingänge werden in der E-Zeile angezeigt.

Die Werte der Analog-Eingänge in den Feldern EX und EY
Die M-Ausgänge können durch Maus-Klick auf L bzw. R bedient werden. Wird gleichzeitig noch die Strg-Taste gedrückt, bleibt der M-Ausgang dauerhaft eingeschaltet. Ausschalten durch Klick auf das zugehörende Mx-Feld.

5. Ebenso kann bei Anzeige Editier-Modus das im Editier-Feld angezeigt Anwendungsprogramm geändert und ergänzt werden. Nach Start der IDE enthält das Editier-Feld ein kleines Beispiel-Programm, das man nach eigenem Bedarf ändern oder auch löschen kann (Neu.VBS). Eigene Anwendungen kann man über das Menü Datei laden und speichern.
6. Ein Klick auf den jetzt mit RUN beschrifteten Button startet die Anwendung. Die Beschriftung des Buttons wechselt in HALT, in der Status-Zeile wird läuft angezeigt (solange es nicht durch die Anwendung überschrieben wird).
7. Vor dem Start der Anwendung können die EA / EB – Felder mit numerischen Werten belegt werden. Die Anwendung kann dann über gleichnamige Funktionen darauf zugreifen.
8. Das Programm kann durch Klick auf den HALT-Button beendet werden. Es gibt Situationen wo das nicht mehr möglich ist (z.B. Do .. Loop – OHNE Until ..), Dann hilft nur noch der Weg über den Task-Manager (Strg+Alt+Entf), vorher kann die Anwendung aber noch gespeichert werden.
9. Während des Programmablaufs können Fehlermeldungen angezeigt werden. Das geschieht in einer MessageBox. Der Fehlertext beginnt mit Fehler in Zeile ... Man sollte ggf. die MessageBox so verschieben, daß man die markierte Zeile im Editierfeld lesen kann. Im Fehlertext folgt eine Nummer und dann die Beschreibung des Fehlers. Die Beschreibung des Fehlers ist meist recht aussagekräftig. Allerdings kann der Grund für die Fehlermeldung durchaus auch an anderer Stelle liegen. Man sollte im Zweifelsfall die Dokumentation zurate ziehen.
10. Die IDE kann normalerweise über den ENDE-Button beendet werden, während des Programmlaufs ist er allerdings deaktiviert, das Programm muß erst über den HALT-Button angehalten werden.

Die IDE greift beim Start einer Anwendung auf die Dateien Global.VBS und Neu.VBS (im Progammverzeichnis) zu. Beide Dateien können – mit Vorsicht - nach eigenen Bedürfnissen geändert werden. Global.VBS enthält Konstanten, die für jede Anwendung gelten (z.B. ftiEin / ftiAus) und Neu.VBS enthält das kleine Beispiel, das bei Start der IDE angezeigt wird, den Inhalt kann man schlicht löschen oder aber auch einen eigenen Standard-Programmrahmen erfinden.

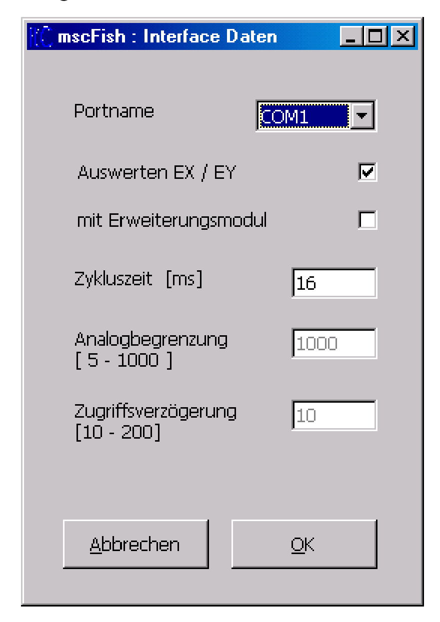
Anschluß des Interfaces

Anschluß je nach Gerät an COM bzw. LPT, Stromversorgung 9V bzw. 6V mit einem Netzteil, das min. 500mA, besser 1000 mA liefert, das Netzteil sollte stabilisiert sein. Die fischertechnik Netzteile erfüllen meist diese Bedingungen. Hinweis : alte (graue) Motoren können auch mit 9V am Intelligent Interface laufen.

Der Active Mode des Intelligent Interface wird nicht unterstützt. Slaves/Extension Modules werden unterstützt, im Interface Panel wird z.Zt. aber nur das Master-Interface angezeigt.

Einstellen der Interfacewerte

Die Werte für den Interfacebetrieb werden über das Menü Extras | Optionen der IDE eingestellt.



Man sollte die Interfacewerte in der Reihenfolge der Form einstellen, also zuerst den Portnamen wählen. Damit verbunden werden für die weiteren Werte Standard-Vorgaben gemacht, die man nur mit Vorsicht ändern sollte.

EX / EY	Slave	Zykluszeit COM	LPT*
x	x	16	100
x	-	16	100
-	x	12	8
-	-	10	8

* Die LPT-Werte sind "vorsichtig" gewählt, sie können unterschritten werden. Mit EX/EY = 20, ohne = 1.

Wenn man nicht vorhat, Analogwerte zu messen, kann man den Haken bei "Auswerten EX / EY deaktivieren. Wird diese Option beim parallelen (Universal) Interface angewählt, so steigt die vorgegebene "Zykluszeit" d.h. das Zeitintervall in dem der Interface Status upgedatet wird, drastisch. Man sollte die Zykluszeit nur mit Vorsicht kleiner wählen, da der Rechner dabei "einfrieren" kann.

Der Erweiterungsmodul (Slave / Extension Module) wird unterstützt, allerdings z.Zt. nicht im Interface Panel angezeigt.

Die Analogbegrenzung beeinflußt beim parallen Interface die angezeigten Analogwerte. Beim LPT1/3 werden sie gestreckt d.h. bei größeren Werten über einen größeren Anzeigebereich gedehnt, Bei LPT ist die Analogbegrenzung eine Obergrenze bei der darüber hinausgehende Werte abgeschnitten werden.

Die Zugriffsverzögerung ist eine Korrektur beim Zugriff auf das Interface bei LPT1/3. Größere Werte für schnellere Rechner.

Einstellen der Programmiersprache

mscFish kann wahlweise mit JavaScript (in der Microsoft-Version JScript) bzw. VBScript betrieben werden. Die Auswahl der Programmiersprache erfolgt über das Menü Extras | Sprachen.



Das ist nur gleich nach dem Programmstart möglich. Außerdem kann ausgewählt werden, ob den FishFace-Befehlen ein ft. vorangestellt werden muß. Sinn voll ist das dann, wenn das mit mscFish erstellte Programm auch außerhalb der mscFish-Umgebung (z.B. innerhalb einer HTML-Seite) laufen soll – siehe Anhang.

Hilfe – Dokumentation

VBScript

Die deutsche Hilfe-Datei **Script56.CHM** von Microsoft enthält im Kapitel VBScript | VBScript-Benutzerhandbuch eine ausführliche Übersicht über die kompletten Möglichkeiten von VBScript und im Kapitel VBScript | VBScript-Sprachverzeichnis eine Referenz aller VBScript-Befehle. Zusätzlich im Kapitel Script-Laufzeit | FileSystemObject eine Beschreibung der umfangreichen Möglichkeiten des Dateizugriffs.

Im nachfolgenden Abschnitt "Einführung in die Programmierung" wird am Ende der Kapitel auf passende Seiten der VBScript-Dokumentation hingewiesen.

Script56.CHM ist Bestandteil von mscFish30Setup. Es kann auch kostenlos von www.microsoft.com/germany/scripting geladen werden (Stand März 2003)

FishFace

Zu FishFace gibt es zwei im Aufbau und Inhalt recht ähnliche Handbücher für JScript/VBScript im PDF-Format. Sie bestehen aus einem **Übersichtsteil**, einer **Einführung in die Programmierung** und einem **Referenzteil** für FishFa30.DLL.

Einführung in die Programmierung

Allgemeines

Im diesem Abschnitt wird eine einfache Einführung in die Programmierung gegeben. Sie ist für Programmieranfänger mit einiger Windows-Erfahrung, aber ohne Programmierkenntnisse gedacht. Die Einführung erfolgt anhand von praktischen Beispielen, auf eine theoretische Unterlegung wird verzichtet. Ebenso auf eine komplette Darstellung der Sprache VBScript. Die Einführung erfolgt bewußt auf der Basis von "prozeduralen" Elementen um die Einstiegsschwelle niedrig zu halten. Eine Nutzung der OO-Elemente von VBScript ist aber problemlos möglich.

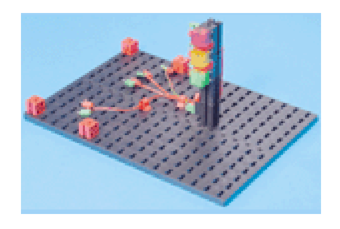
Die Kapitel bauen aufeinander auf und nehmen im Schwierigkeitsgrad zu, sie sollten deswegen nacheinander durchgearbeitet werden.

Bei weitergehendem Interesse kann nahtlos zu der in ausreichendem Maße vorhandenen Computerliteratur gegegriffen werden. Auf die bei Microsoft kostenlos erhältliche VBScript-Dokumentation (s.o.) wird immer wieder am Ende eines Kapitels hingewiesen, sie sollte parallel zum Durcharbeiten dieses Abschnitts zu Rate gezogen werden.

Die Programmbeispiele beziehen sich alle auf Modelle des fischertechnik Kastens "Computing Starter" No. 16 553. Wenn man bereits einige fischertechnik Teile (auch "graue") einschl. fischertechnik Interface besitzt, kann man die Modelle auch leicht ohne den Kasten nachbauen, dann wird empfohlen die "Computing Starter – Bauanleitung" No. 30 434, Preis 7,70 € bei www.knobloch-gmbh.de zu kaufen. Dort sind auch evtl. fehlende Teile erhältlich.

Erste Befehle

Lampen



Lampen an M1 – M3
grün – gelb – rot

```
SetMotor ftiM1, ftiEin
Pause    1000
SetMotor ftiM2, ftiEin
Pause    1000
SetMotor ftiM3, ftiEin
Pause    1000
```

1. Aufbau des Modells mit den Lampen an M1 – M3 des Interfaces.
2. Anschluß des Interfaces an den Rechner, Netzteil anschließen
3. Aufruf von mscFish, den Interface-Anschluß (COM1 ...) kontrollieren, START Klicken.
   Der Punkt neben der Anschluß-Auswahl muß grün werden.
4. Am Interface Panel der Reihe nach mit der Maus auf die Ls klicken.
   Die entsprechende Lampe muß leuchten.

5. Den Text im Editier-Feld löschen und die oben angegebenen Befehle in das Editier-Feld eingeben.Nochmal kontrollieren.
6. RUN klicken.
   Wenn denn doch etwas falsch eingegeben wurde, kommt eine entsprechende Fehlermeldung und die "schuldige" Zeile wird markiert – korrigieren – RUN

Was passiert : die Lampen gehen - eine nach der anderen – im Abstand von einer Sekunde an.

Verwendet wurden dazu zwei **Befehle** `SetMotor` zum Einschalten der Lampen und `Pause` zum Anhalten des Programms.
SetMotor hat zwei **Parameter** die **symbolischen Konstanten** `ftiM1` und `ftiEin`. Der erste steht für den Anschluß (M1 ...), der zweite für die Aktion (Ein, Aus, Links/Rechts drehen). Bei Lampen reicht ein schlichtes `ftiEin`.
Pause hat als Parameter eine **numerische Konstante** (eine Zahl), die angibt, wie lange das Programm anzuhalten ist und zwar in Millisekunden. Die `1000` hier steht also für 1 Sekunde halten.
Das wars denn auch schon.

FishFace : Eine genaue Beschreibung der Befehle ist im Referenzteil zu finden.

Schleife

Das mit den Lampen war ja ganz schön, aber auch schön schnell zu Ende. Man sollte das in einer Schleife (Schweizer und Österreicher : Schlaufe) wiederholen. Dafür gibt es den Befehl Do .... Loop Finish
mit dem man eine Folge von anderen Befehlen "einrahmen" kann um sie zu wiederholen :

```
Do
  SetMotor ftiM1, ftiEin
  Pause    1000
  SetMotor ftiM2, ftiEin
  Pause    1000
  SetMotor ftiM3, ftiEin
  Pause    1000
  ClearMotors
  Pause    1000
Loop Until Finish
```

Die Schleife läuft solange (**Until**), bis die Finish-**Bedingung** True (wahr) wird, d.h. bis in der IDE auf den HALT-Button oder auf der Tastatur auf die Esc-Taste gedrückt wird, man könnte auch bei Finish noch einen Taster angeben : `Finish(ftiE8)`, dann würde die Schleife auch wenn der Eingang E8 am Interface True ist (weil z.B. ein Taster gedrückt wurde) abgebrochen.
Anstelle von Finish können auch andere Bedingungen angegeben werden ... das kommt später.

Wenn man sich den Code (die Befehle) genau ansieht, wird man zwei zusätzliche entdecken. Pause ist bekannt, `ClearMotors` löscht alle M-Ausgänge (M1 – M4). Wenn man sie wegläßt, rührt sich ab dem zweiten Durchlauf gar nichts mehr in der Schleife, alle Lampen werden ständig wieder eingeschaltet, obwohl sie ja schon an sind. Mit ClearMotors ändert sich das und damit mans auch merkt die Pause.

VBScript Index Suchbegriff : Do..Loop-Anweisung
FishFace Referenz : Finish

Schönheit – Lesbarkeit

```
' --- Ampel1.VBS : Eine einfache Ampel ----------------
Const mGruen = 1, mGelb = 2, mRot = 3
Const cLangePause = 1000, cKurzePause = 300

Do
  SetMotor mGruen, ftiEin
  Pause    cLangePause
  SetMotor mGruen, ftiAus
  SetMotor mGelb,  ftiEin
  Pause    cKurzePause
  SetMotor mGelb,  ftiAus
  SetMotor mRot,   ftiEin
  Pause    cLangePause
  SetMotor mGelb,  ftiEin
  Pause    cKurzePause
  SetMotor mRot,   ftiAus
  SetMotor mGelb,  ftiAus
Loop Until Finish
```

Man kann anstelle der allgemeinen symbolischen Konstanten, die die Anschlüsse am Interface bezeichnen, eigene Konstanten einführen, die die am Interface angeschlossenen Geräte bezeichnen. Hier sind das die verschiedenfarbigen Lampen mGruen, mGelb, mRot. Und wenn man schon dabei ist auch noch cLangePause und cKurzePause für den Pause Parameter. So kann man leicht die Einschaltdauer zentral verändern. Die kleinen Buchstaben vor den Konstanten stehen für die Bedeutung der Konstanten (m = M-Ausgang des Interfaces, c = allgemeine Konstante, e stände dann für die E-Eingänge). Das ist eine verbreitete Sitte, deren jeweilige Form mit großem Nachdruck vom Anwender vertreten wird, diese Form ist eine Marotte des Autors.

Groß- und Kleinschreibung ist ebenfalls eine viel "diskutierte" Möglichkeit, den Code lesbarer zu machen. Hier wird konsequent die Kamel-Schreibweise (mit Höckern durch weitere Großbuchstaben im Wort) angewendet. VBS schluckt jede Schreibweise (ist nicht case-sensitiv). Man sollte aber bei einer einheitlichen Schreibweise bleiben.

Üblicherweise wird der Code in Schleifen – ebenfalls aus Gründen der besseren Lesbarkeit – eingerückt.

Zusätzlich sind noch **Kommentare** möglich. Sie beginnen mit einem Hochkomma ('), der Rest der Zeile wird dann von VBS nicht mehr beachtet. Hier wurde ein Kommentar als Programmüberschrift verwendet. Man kann auch Kommentar auf den Rest einer Befehlszeile schreiben.

VBS ist eine zeilenorientierte Programmiersprache. D.h. Befehle mitsamt ihren Parametern müssen auf einer Zeile stehen. Ggf. kann eine Fortsetzungszeile eingeführt werden : am Zeilenende ein Leerzeichen (blank) und eine Underscore (_) und dann weiter auf der nächsten Zeile. Andere Sprachen verwenden ein Semikolon (;) zur Begrenzung eines Befehls. Beides hat seine Vor- und Nachteile.

VBScript Index : Const-Anweisung

Variable

```
' --- Ampel1.VBS : Eine einfache Ampel ----------------
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3
Dim LangePause, KurzePause

LangePause = 1000
KurzePause = LangePause/4
Do
  SetMotor mGruen, ftiEin
  Pause    LangePause
  SetMotor mGruen, ftiAus
  SetMotor mGelb,  ftiEin
  Pause    KurzePause
  SetMotor mGelb,  ftiAus
  SetMotor mRot,   ftiEin
  Pause    LangePause
  SetMotor mGelb,  ftiEin
  Pause    KurzePause
  SetMotor mRot,   ftiAus
  SetMotor mGelb,  ftiAus
Loop Until Finish
```

Konstanten enthalten einen festen Wert, Variable können einen veränderlichen (variablen) Wert enthalten. Durch Änderung von `Const cLangePause = 1000` des vorhergehenden Beispiels in `Dim LangePause` erhält man eine Variable der man noch einen Anfangswert zuweisen muß : `LangePause = 1000`, (standard ist 0) wie gehabt als Konstante. Der Vorteil von Variablen ist, dass man diesen Wert wieder ändern kann, das auch mehr oder weniger kompliziert durch **Ausdrücke** : `KurzePause = LangePause / 4`. Der Ausdruck (LangePause / 4) wird hier einer weiteren Variablen (KurzePause) **zugewiesen** d.h. der Wert der KurzePause beträgt ¼ der LangePause. Man kann natürlich noch viel komplexere Ausdrücke bilden. U.a. kann man alle **Operatoren** (+, -, *, /, ()) der Arithmetik einsetzen.
In Ausdrücken können Variablen, Konstante und Funktionen (kommt später) eingesetzt werden. Im Beispiel oben waren es die Variable LangePause und die Konstante 4.
(50 + 100) * 2 –50 ist ebenfalls ein gültiger Ausdruck und ergibt wieder 250.

Die Steueranweisung **`Option Explicit`** sollte stets als erste Anweisung (ggf. nach Kommentaren) im Programm stehen. Sie weist VBS an nur Variablen (bzw. Konstanten) zu akzeptieren, die vorher durch einen Dim / Const Befehl angemeldet (deklariert) wurden. Es ginge auch ohne, in diesem Fall würden Schreibefehler bei einer Variablen, z.B. LangePausen (statt LangePause) zum Anlegen einer weiteren Variablen mit dem Anfangswert 0 führen. D.h. hieße, daß die oben so schön auf 1000 initialisierte (mit dem Wert 1000 besetzte) Variable LangePause im Befehl `Pause LangePausen` zu einer Pause der Länge 0 führen würde, d.h. Lampe ist gleich wieder aus. Macht man den Schreibfehler schon bei KurzPause = LangePausen/4, so sind alle KuzePause = 0 und die gelbe Lampe leuchtet nicht mehr wahrnehmbar. Da wird man recht heftig ins Grüblen kommen (Schon weil man solche krummen Sachen eher der IDE als sich selber zutraut. Ist ja schließlich "alles" richtig!).

VBScript Inhalt : VBScript-Benutzerhandbuch | Grundlagen | Variablen .... Datentypen
VBScript Index : Option Explicit

Ein- und Ausgaben

```
' --- Ampel1.VBS : Eine einfache Ampel ---------------
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3
Dim LangePause, KurzePause, Runde

LangePause = 1000*EA
KurzePause = LangePause/4
PrintStatus "--- Die Ampel bei der Arbeit ---"
Do
  Runde = Runde + 1
  PrintLog "Runde : " & Runde
  SetMotor mGruen, ftiEin
  Pause    LangePause
  .........
  SetMotor mGelb,  ftiAus
Loop Until Finish
```

In der neuen Fassung wird über die Befehl **EA** auf das korrespondierende Feld der IDE zugegriffen und ein Faktor zu Modifikation der Ablaufzeiten ausgelesen (Vorgabe ist 1). So kann man recht elegant die bisherige "auf Tempo" getrimmte Fassung besser den realen Bedingungen an einer Ampel anpassen (bei einem Faktor 30 muß man dann genauso endlos warten wie an einer realen Ampel).

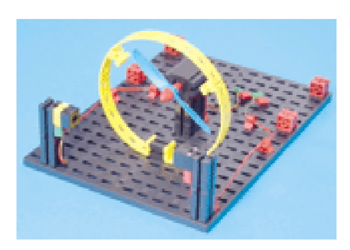
Mit dem Befehl **PrintStatus** kann in die hellblaue Statuszeile geschrieben werden. Hier wird die **Textkonstante** "--- Die Ampel bei der Arbeit ---" ausgegeben. Das ist eine neue Art von Konstante – bisher waren nur numerische Konstanten im Spiel. Textkonstanten beginnen und enden mit einem doppelten Hochkomma (") und können alle anzeigbaren Zeichen enthalten, sie müssen auf eine Zeile passen.

Die neue Variable **Runde** soll die durchlaufenen Schleifen zählen. Deklaration oben mit Dim wie gewohnt, keine Besetzung mit einem Anfangswert, hier wird der Standardwert 0 genutzt. Zu Beginn jeder Schleife wird dann der Wert von Runde um 1 erhöht
Der Wert von Runde + 1 ist ein Ausdruck, der wieder Runde zugewiesen wird, dadurch wird der alte Wert von Runde (beim ersten Mal 0), der noch im Ausdruck galt, überschrieben.

PrintLog schreibt dann in den bisher noch nicht genutzten Log-Bereich. Geschrieben wird eine Text-Konstante und eine Zahl (der aktuelle Wert von Runde) die durch den Operator Ampersand (&) mit einander in geeigneter Weise zu Text verbunden werden. Im Log-Bereich werden die Ausgaben nicht überschrieben, sondern gescrollt (nach oben geschoben), die jeweils letzte Zeile wird markiert und bleibt im Fenster sichtbar. Über den Scrollbar rechts können auch die anderen Zeilen angezeigt werden.

FishFace Referenz : PrintStatus, PrintLog

Motoren, Taster und Lichtschranken



Motor an M1,
Lichtschranke M2 Lampe,
E1 Phototransistor

```
Const mMotor = 1, mLampe = 2
Const eTaster = 1, ePhoto = 1
Do
  SetMotor    mMotor, ftiEin
  Pause 5000
  SetMotor    mMotor, ftiAus
Loop Until Finish
```

"Der Händetrockner soll nun so programmiert werden, daß, sobald die Lichtschranke unterbrochen wird, der Lüfter ein- und nach 5 Sekunden wieder ausgeschaltet wird."

Mit dem oben angegebenen Programm läuft der Trockner, aber leider trotz Pause ewig. Was fehlt ist ein Auslöser : die Lichtschranke oder ein einfacher Taster, erstmal wird ein Taster an E1 probiert.

```
Do
  If GetInput(eTaster) Then
    SetMotor    mMotor, ftiEin
    Pause 5000
    SetMotor    mMotor, ftiAus
  End If
Loop Until Finish
```

Als erstes fällt die neue Konstruktion `If ... Then ... End IF` auf. Sie ist, wie das schon bekannte Do .. Loop, eine Programmklammer. Hier werden die Befehle zum Schalten des Trocknermotors eingeklammert. Dem If folgt eine Bedingung (ganz wie beim Loop Until), hier ist das GetInput(eTaster). Dieser Befehl fragt den Eingang E1 ab, wenn der geschlossen ist, liefert GetInput das Ergebnis True (Wahr). D.h. der Inhalt der Klammer wird unter der Bedingung ausgeführt, daß der Eingang E1 True ist.
Wenn man nun kurz auf den **Taster** (Schließer : angeschlossen sind die Kontakte 1 und 3) drückt, läuft der Motor an und tut das weitere 5 Sekunden und wird dann wieder abgeschaltet. D.h. das gesamte Programm wartet hier, ein erneutes Drücken des Taster hat in dieser Zeit keine Wirkung.

Der Taster ist recht unbequem, deswegen nun zur Lichtschranke.

```
SetMotor mLampe, ftiEin
Pause 1000
Do
  If GetInput(ePhoto) Then
    SetMotor    mMotor, ftiEin
    Pause 5000
    SetMotor    mMotor, ftiAus
  End If
Loop Until Finish
```

Vor dem Do-Loop muß erstmal die Lichtschranke aktiviert werden (jetzt ist der Phototransistor an E1 angeschlossen). Also SetMotor mLampe, ftiEin und ein wenig warten, damit sich der Phototransistor an das Licht gewöhnt, man kann das auf dem Interface Panel kontrollieren. Und dann die If Bedingung in GetInput(ePhoto) ändern. Der Motor läuft, nur leider auch schon, wenn die Lichtschranke noch gar nicht unterbrochen ist. Wenn man dann die Hand reinhält, bleibt der Motor nach spätestens 5 Sekunden stehen, es läuft also genau umgekehrt. Daran muß gedreht werden :

```
If Not GetInput(ePhoto) Then
```

stellt die Bedingung auf den Kopf. Das If wird jetzt ausgeführt, wenn GetInput(ePhoto) False liefert, die Lichtschranke also unterbrochen ist. Verursacht wird das durch das vorangestellte **NOT**. Jetzt geht's!

VBScript Inhalt : Benutzerhandbuch | Grundlagen | Verwenden von Bedingungsanweisungen
VBScript Index : If

Jetzt noch einige Verzierungen :

```
' --- Trockner.VBS : Händetrockner ---
Option Explicit
Const mMotor = 1, mLampe = 2
Const eTaster = 1, ePhoto = 1
Dim Kunden

Kunden = 0
SetMotor mLampe, ftiEin
Pause 1000
Do
  PrintStatus "Trockner bereit"
  If Not GetInput(ePhoto) Then
    PrintStatus "Trockner läuft"
    Kunden = Kunden + 1
    PrintLog "Kunde Nr : " & Kunden & " am " & Now
    SetMotor    mMotor, ftiEin
    Pause 5000
    SetMotor    mMotor, ftiAus
  End If
Loop Until Finish
```

Mit PrintStatus wird in der blauen Statuszeile angezeigt, ob der "**Trockner bereit**" ist oder ob der "**Trockner läuft**".

Außerdem wird in im Log-Bereich die Nutzung des Trockner protokolliert. Dazu wird eine Variable **Kunden** angelegt und vor dem Do-Loop auf 0 initialisiert. Jedesmal, wenn der Motor gestartet werden soll wird **Kunden + 1** hochgezählt und mit PrintLog ausgegeben. Die Ausgabe wird dabei ganz raffiniert "zusammengebastelt" :
Zuerst die **Textkonstante** "Kunde Nr : ", dann, durch & verknüpft, die aktuelle Kundenzahl, dann wieder eine Textkonstante " am " und zum Schluß **Now**, das Tagesdatum mit Uhrzeit, das vom System bereitgestellt wird.

Das wars.

Und noch eine Spielerei

```
    SetMotor mMotor, ftiEin
    Pause 3000
    SetMotor mMotor, ftiEin, ftiHalf
    Pause 2000
    SetMotor mMotor, ftiAus
```

Nach dem gewohnten SetMotor kommt noch eins mit einem zusätzlichen Parameter **ftiHalf**. Bedeutet : der Motor läuft jetzt nur noch mit halber Drehzahl (so zum Nachtrocknen). Der Befehl SetMotor hat also einen Parameter mehr als man dachte. Das funktioniert so : eigentlich hat er immer drei, wenn man den dritten wegläßt, wird das intern erkannt und er wird durch einen default (Ersatz) Wert (hier ftiFull) ersetzt. Erlaubt sind hier auch die Werte 0 – 15 für aus über 7 für ftiHalf und 15 für ftiFull. Man kann also noch mehr spielen.

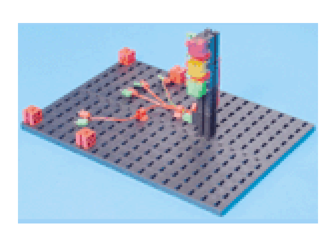
Und noch ein Nachtrag

Wenn an einem M-Ausgang ein Motor angeschlossen ist, kann der Ausgang auch mit **ftiLinks** (= ftiEin) und **ftiRechts** betrieben werden, es kann also die Drehrichtung des Motors vorgegeben werden (also der Langsamgang vielleicht ftiRechts – rum?).

Zur Drehrichtung : mit ftiLinks wird die Drehrichtung bezeichnet, die der Motor einnimmt, wenn man beim Interface Panel auf "L" drückt. Wenn das falschrum ist, sollte man am Motor das Kabel umstecken. Am Interface selber sollt der rote Stecker immer in "der ersten Reihe" (also vorn) stecken, das schafft Übersicht.

VBScript Inhalt : VBScript-Operatoren

Intermezzo : Nocheinmal die Ampel



Lampen an M1 – M4
grün – gelb – rot –
FußGrün (an der Seite)
Taster (1/3) an E1 und E2

```
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3,
Const mFuss = 4
Const eFussWunsch = 1
Do
  SetMotor mGruen, ftiEin
  If GetInput(eFussWunsch) Then
    Pause    1000*EA
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    Pause    1000*EA
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  End If
Loop Until Finish
```

Das Programm ist weitgehend bekannt, aber eine If ... End If "Klammer" ist hinzu gekommen und schon ist daraus eine Fußgängerampel geworden. GetInput(eFussWunsch) wartet auf das Drücken der Tasters zur Anforderung einer Fußgängerphase, erst dann läuft es mit mGelb weiter. eFussWunsch wurde vor der vorhanden Pause platziert um zu verhindern, daß sofort nach Ablauf einer Fußphase gleich wieder eine neue gestartet wird.

Was noch fehlt ist eine Fußgängerampel, da es am Interface ein wenig eng wird, nur eine grüne an M4. Das sieht dann so aus :

```
Do
  SetMotor mGruen, ftiEin
  If GetInput(eFussWunsch) Then
    Pause    1000*EA
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    SetMotor mFuss,  ftiEin
    Pause    1000*EA
    SetMotor mGelb,  ftiEin
    SetMotor mFuss,  ftiAus
    Pause    250*EA
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  End If
Loop Until Finish
```

ElseIf und Subs

Nachts werden Ampeln oft auf Gelb-Blinken umgestellt :

```
Const eFussWunsch = 1, eGelbBlinken = 2
Do
  If GetInput(eGelbBlinken) Then
    Blinken
  ElseIf GetInput(eFussWunsch) Then
    Pause    1000*EA
    SetMotor mGruen, ftiAus
```

```
    .....
    SetMotor mGelb,  ftiAus
    SetMotor mGruen, ftiEin
  Else
    SetMotor mGruen, ftiEin
  End If
Loop Until Finish

Sub Blinken
  SetMotor mGelb, ftiEin
  Pause 500*EA
  SetMotor mGelb, ftiAus
  Pause 400*EA
End Sub
```

Das GelbBlinken wird über die Abfrage eines weiteren Tasters eGelbBlinken geschaltet. Die vorhandene If ... End If Klammer wird um eine neue Klausel – ein **ElseIf** – erweitert. Beim If wird wie bisher auf eGelbBlinken getestet, wenn das nicht zutrifft (eGelbBlinken hat Vorrang) auf eFussWunsch und wenn das auch nichts war wird ganz normal das (Auto)mGruen eingeschaltet (wird jetzt also nicht mehr bei jedem Schleifendurchlauf geschaltet).

Das Blinken selber wird durch ein ebenfalls neues Sprachelement – ein **Unterprogramm** (Sub) – gesteuert. In einem Unterprogramm werden logisch zusammengehörenden Befehle zusammengefaßt und außerhalb des normalen Programmablaufs abgestellt. Ein Unterprogramm wird durch seinen Namen aufgerufen (beim If eGelbBlinken : **Blinken**). Vorteil eines Unterprogramms : das eigentliche (Haupt)Programm wird übersichtlicher, bei mehrmaligem Aufruf (wenn man auch noch woanders Blinken will) wird vorhander Code wiederverwendet.

VBScript Index : If ..Then..Else-Anweisung

SetMotors

Die vielen SetMotor Ein und Aus sind langsam unübersichtlich, mit SetMotor**s** kann man sie zusammenfassen und alle Lampen auf einmal schalten :

```
' --- FussAmpel3.VBS : Eine einfache Ampel --------
Option Explicit
Const mGruen = &H1, mGelb = &H4, mRot = &H10, mFuss = &H40
Const eFussWunsch = 1, eGelbBlinken = 2
Do
  If GetInput(eGelbBlinken) Then
    Blinken
  ElseIf GetInput(eFussWunsch) Then
    Pause     1000*EA
    SetMotors mGelb
    Pause     250*EA
    SetMotors mRot + mFuss
    Pause     1000*EA
    SetMotors mGelb
    Pause     250*EA
    SetMotors mGruen
  Else
    SetMotors mGruen
  End If
Loop Until Finish

Sub Blinken
  SetMotors mGelb
  Pause     500*EA
  ClearMotors
  Pause 400
```

```
End Sub
```

Dazu muß man wissen, das der Status aller M-Ausgänge in einem OutputStatusword abgebildet werden kann, jeweils zwei bit für einen M-Ausgang : `00 00 00 00 die` M-Ausgänge M4 – M1 sind abgeschaltet. Es wird mit SetMotor**s** an das Interface weitergegeben.

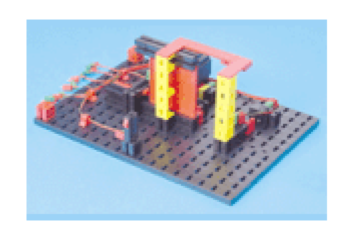
```
00 01 00 00        (&H10, mRot  in binär Darstellung)
01 00 00 00 +      (&H40, mFuss)
-------------
01 01 00 00 =     mFuss ein, mRot ein, mGelb aus, mGruen aus
```

Dazu wurden die Lampen-Konstanten entsprechend geändert. Sie enthalten jetzt nicht mehr die Nummer des M-Ausganges sondern die bit-Position des M-Ausganges im OutputStatusword. Geschrieben wurde das in der kürzeren Hexa-Darstellung, möglich wäre auch die dezimale Darstellung 1, 4, 16 und 64, die interne Darstellung ist immer binär.

Wenn man das Programm noch verschönern will, kann man die PrintStatus- und PrintLog-Ausgaben des Händetrockners entsprechend modifiziert übernehmen.

FishFace Referenz : SetMotors, ClearMotors
VBScript Inhalt : VBScript-Prozeduren
VBScript Index : Subanweisungen

Über das Türenschieben



Motor an M1
Lichtschranke M2, E4
Taster TürAuf E2
Taster TürZu E1
Taster Öffnen E3

"Wenn Taster E3 gedrückt wird, soll sich die Tür öffnen und nach fünf Sekunden wieder schließen."

```
' --- Tuer.VBS : Schiebetür ---
Option Explicit
Const mTuer = 1, mLampe = 2
Const eTuerZu = 1, eTuerAuf = 2, eOeffnen = 3
Const ePhoto = 4
Do
  SetMotor mTuer, ftiLinks
  WaitForInput eTuerZu, False
  SetMotor mTuer, ftiAus

  WaitForInput eOeffnen

  SetMotor mTuer, ftiRechts
  WaitForInput eTuerAuf
  SetMotor mTuer, ftiAus
  Pause 5000
Loop Until Finish
```

Zunächstmal wird die Tür geschlossen (SetMotor). Das wird durch den neuen Befehl **WaitForInput** überwacht. WaitForInput überwacht den eTuerZu-Eingang am Interface und zwar darauf, daß der zugehörende Taster öffnet (deswegen der Parameter False). Das ist zunächst etwas überraschend, wenn man sich aber die baulichen Verhältnisse und die Vorliebe für eine Schaltung als Schließer (Kontakte 1 und 3) ansieht, verständlich.

Und dann wird schon wieder gewartet : auf eine Anforderung zum Türöffnen über eOeffnen. Der Taster ist wieder als Schließer geschaltet und wird gedrückt (geschlossen) deswegen wird hier auf `WaitForInput eOeffnen, True` gewartet, da True der default Parameter ist, kann man ihn auch weglassen, also `WaitForInput eOeffnen`.

Und dann geht endlich die Tür für 5 Sekunden auf.

FishFace Referenz : WaitForInput

Überwachung durch Lichtschranke

```
SetMotor mLampe, ftiEin
Pause 1000
WaitForInput ePhoto, True
Do
  Do While GetInput(eTuerZu)
    SetMotor mTuer, ftiLinks
    If Not GetInput(ePhoto) Then TuerOeffnen
  Loop
  SetMotor mTuer, ftiAus

  If GetInput(eOeffnen) Or GetInput(ePhoto) = False Then TuerOeffnen

Loop Until Finish

Sub TuerOeffnen
  SetMotor mTuer, ftiRechts
  WaitForInput eTuerAuf
  SetMotor mTuer, ftiAus
  Pause 5000
End Sub
```

Das Modell sieht sie ja vor, die Lichtschranke ePhoto – mLampe, nun wird sie auch genutzt. Das fängt, wie vom Händetrockner gewohnt, mit dem "Anwärmen" der Lichtschranke an. Dann wird noch gewartet, dass sie auch wirklich OK ist, dann erst geht's mit dem gewohnten Do – Loop los.

Es folgt dann aber gleich noch einer, aber ein ungewohnter.
`Do While GetInput(eTuerZu)`. Neu ist das While, die Schleife wird solange durchlaufen, wie die Bedingung eTuerZu zutrifft (d.h. ja eigentlich Tuer in noch nicht ganz zu, s.o.), außerdem ist dies eine "abweisende Schleife", wenn die Bedingung nicht zutrifft, d.h. die Tür bereits geschlossen ist, wird sie nicht durchlaufen (die bisherigen – nicht abweisenden – Schleifen wurden mindestens einmal durchlaufen). Das hat hier den Vorteil, daß nicht ständig an der Tür "gerüttelt" wird.

In der Schleife wird dann der Türmotor eingeschaltet (Richtung Schließen) das wird aber gleich wieder revidiert, wenn irgendetwas in die Lichtschranke (GetInput(ePhoto) geraten ist, dann wird das Unterprogramm TuerOeffnen ausgeführt. Nach der Schleife wird der Türmotor wieder ausgeschaltet.

Das nachfolgende If ersetzt das bisherige WaitForInput. Jetzt werden der eOeffnen-Taster und die Lichtschranke auf eine Anforderung zum TürÖffnen abgefragt. Im positiven Fall : TuerOeffnen. Hier also ein Beispiel für die mehrmalige Verwendung eines Unterprogramms.

```
Sub TuerOeffnen(Normal)
  PrintStatus "--- Tür öffnet ---"
  If Normal Then PrintLog "Tür wurde geöffnet : " & Now _
  Else PrintLog "Tür-Zwischenfall   : " & Now
  SetMotor mTuer, ftiRechts
  WaitForInput eTuerAuf
  SetMotor mTuer, ftiAus
  PrintStatus "--- Tür geöffnet ---"
  Pause 5000
End Sub
```

Man kann einem Unterprogramm auch einen, oder mehrere Parameter mitgeben um den Ablauf des Unterprogramms zu steuern. Dazu muß sie (die Parameter) in der Sub Definitionszeile angeführt werden (formale Parameter, hier **Normal**), sie können dann im Unterprogramm wie normale Variable genutzt werden, hier wird unterschieden, ob die Tür per Anforderung oder durch "Kiste in der Lichtschranke" geöffnet wurde. Beim Aufruf des Unterprogramms muß dann ein entsprechender aktueller Parameter mitgegeben werden : `TuerOeffnen False` bei der "NotÖffnung" und `TuerOeffnen True` nach Anforderung.

VBScript Index : Or , Subanweisung

Temperatur-Regelung



Motor an M1
Lampe an M2
NTC-Widerstand an EX

"Oberhalb einer bestimmten Temperatur schaltet die Heizung aus- und die Kühlung ein, bei Erreichen eines unteren Grenzwertes soll die Heizung ein- und die Kühlung ausgeschaltet werden."

```
Option Explicit
Const mMotor = 1, mLampe = 2
Const aNTC = 0
Dim Temperatur
Do
  Temperatur = GetAnalog(aNTC)
  If Temperatur < EA Then
    SetMotor mLampe, ftiAus
    SetMotor mMotor, ftiEin
  ElseIf Temperatur > EB Then
    SetMotor mLampe, ftiEin
    SetMotor mMotor, ftiAus
  End If
Loop Until Finish
```

Als erstes wird in der Do – Loop Schleife die Variable Temperatur mit dem aktuellen Wert des NTC (Negative Temperature Coefficient, Widerstand, der bei steigender Temperatur kleinere Werte annimmt, also nicht parallel zur Temperatur steigt – deswegen Negative) besetzt. Die Werte können unterschiedlich ausfallen, man sollte anfangs ein wenig experimentieren (der NTC – zwischen Daumen und Zeigefinger genommen – erwärmt sich rapide) und die Werte bei EX auf dem Interface Panel ablesen.

Danach treffen wir wieder auf eine If ... ElseIf ... End If Konstruktion. Wenn der Temperaturwert kleiner als der untere Wert (EA) ist, dann ist es zu heiß (-> NTC), es wird gekühlt. Wenn dann der Temperaturwert größer als EB ist, wird wieder geheizt. Dies Programm wurde mit EA = 575 und EB = 600 getestet.

FishFace Referenz : GetAnalog

HINWEIS : Beim parallelen (Universal) Interface ist hier von der Standard-Einstellung des Interfaces (Menü Extras | Optionen) abzuweichen. "Auswerten EX / EY " muß angekreuzt werden. Die vorgegebene Zykluszeit von 100 ms sollte erstmal beibehalten werden. Man sollte aber mit dem Wert experimentieren. Diese drastisch höhere Zykluszeit (Standard ohne = 8) verändert das Anwortverhalten des Interfaces. Alternativ kann mit **`GetAnalogDirect`** gearbeitet werden, dann kann die Einstellung "ohne" mit Zykluszeit 8 beibehalten werden. Der Analogwert wird dann direkt vom Interface gelesen, das Programm hält dann an dieser Stelle ca. 100 ms.

Dreipunkt-Regelung

Bei dem vorherigen Beispiel war immer etwas los (Kühlen / Heizen im Wechsel) und trotzdem wurde die Temperatur im Versuch nur im Bereich von 575 bis 600 gehalten, könnte man das nicht auch einfach durch "Abwarten" erreichen. Also abschalten und warten bis die Temperatur – je nach Lage der Dinge – von alleine wieder steigt bzw. fällt. Das würde dann auch noch Energie sparen :

```
' --- TemperaDreiTT.VBS : Temperaturregelung ---
Option Explicit
Const mMotor = 1, mLampe = 2
Const aNTC = 0
Dim UT, OT
UT = EA * (1 - EB / 100)
OT = EA * (1 + EB / 100)
PrintLog "Zieltemperatur : " & EA
PrintLog "Untergrenze    : " & UT
PrintLog "Obergrenze     : " & OT
Do
  If Temperatur < UT Then
    PrintStatus "--- Heizt : " & Temperatur & " ---"
    SetMotor mLampe, ftiEin
    SetMotor mMotor, ftiAus
    UT = EA
  ElseIf Temperatur < OT Then
    PrintStatus "--- Temperatur : " & Temperatur & " ---"
    ClearMotors
    UT = EA * (1 - EB / 100)
    OT = EA * (1 + EB / 100)
  Else
    PrintStatus "--- Kühlt : " & Temperatur & " ---"
    SetMotor mLampe, ftiAus
    SetMotor mMotor, ftiEin
    OT = EA
  End If
Loop Until Finish

Function Temperatur
  Temperatur = (1000 - GetAnalog(aNTC)) / 10 - 12
End Function
```

Hier wurden mehrere Maßnahmen in einen Schritt in ein (beinahe) neues Programm eingebaut :

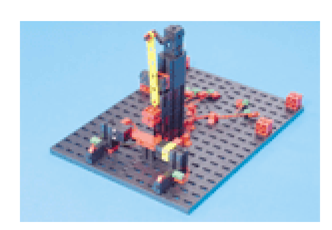
1. **Zieltemperatur** : In EA wird jetzt eine echte Temperatur angegeben, die als Mittelwert zu halten ist. Die vom Mittelwert erlaubten Abweichungen werden in EB in Prozent angegeben. Im Beispiel sind das EA = 29 (geschätzte Grad) und EB = 2%
2. **Function** Temperatur : die mit GetAnalog gemessenen Werte im Bereich von 0 – 1000 werden in Grad umgerechnet. Dazu wird ein neues Konstrukt, eine Function, genutzt. Eine Function ist ein Unterprogramm, das einen Wert als Ergebnis seines Aufrufs zurückgibt. Hier ist das die aktuelle Temperatur. Der gemessene Wert (GetAnalog) wird erstmal umgedreht (1000 – eNTC, jetzt entspricht ein größerer Wert auch einer höheren Temperatur. Der ermittelte Wert wird durch 10 geteilt und dann werden nochmal 12 abgzogen, das sollen dann Grad Celsius sein. Der NTC wurde nicht wirklich geeicht, aber die Werte sind realitätsnah im Bereich 25 – 35.
3. Die Variablen **UT** und **OT** stehen für untere/obere zulässige Temperatur. Sie werden aus Zieltemperatur EA und prozentualer Abweichung EB errechnet und gleich auch im Log-Feld ausgegeben, weils doch mit dem Kopfrechnen so seine Probleme hat.

4. Die If ... ElseIf ... End If Konstruktion wurde um einen Else Zweig erweitert, die zugehörenden Temperatur-Abfragen wurden geändert und den neuen Temperaturwerten angepaßt :
   If : **Zu niedrige Temperatur**
   ElseIf : **Temperatur zwischen UT und OT** also im erlaubten Zielbereich
   Else : **Zu hohe Temperatur**
5. Geschaltet wird wie bisher, aber UT und OT werden laufend geändert. Motto : **Wenn schon Kühlen/Heizen, dann aber richtig und dann Pause**. Wird eine niedere Temperatur erkannt wird UT = EA gesetzt um ein höheres Aufheizen bis zur Zieltemperatur zu erreichen. Beim Kühlen wird dann entsprechend OT auf EA abgesenkt. Bei Temperaturen im Zielbereich werden UT / OT wieder auf die Ausgangswerte gesetzt.
6. In der **StatusZeile** wird die aktuelle Temperatur angezeigt verbunden mit dem Hinweis : Heizt, Temperatur, Kühlt.

Ganz schön viel Stoff für so ein doch noch eher kleines Programm. Tip : Wenn einseitig nur geheizt und pausiert wird. Daumen und Zeigefinger um den NTC führen ihn zu beachtlichen Temperaturen und der Wind kommt.

VBScript Index : Function-Anweisung

Stanzmaschine



Motor an M1
Lampe an M2
Endtaster an E1
Photo-Widerstand an E2
Bedientaster links an E3
Bedientaster rechts an E4

"Die Maschine soll ein Teil in einem Arbeitsgang mit 4 Hüben stanzen. Sie darf nur starten, wenn der Bediener beide Taster betätigt und gleichzeitig die Lichtschranke geschlossen ist. Eine Unterbrechung der Lichtschranke während eines Arbeitsgangs stoppt die Maschine mit Warnsignal."

```
' --- StanzMaschine.VBS ---
Option Explicit
Const mStanze = 1, mLampe = 2
Const eEnde = 1,   ePhoto = 2, eLinks = 3,
eRechts = 4
Dim Produktion, TeilOK, i

Produktion = 0
SetMotor mLampe,  ftiEin
SetMotor mStanze, ftiEin
WaitForInput      eEnde
SetMotor mStanze, ftiAus

Do
  If GetInput(ePhoto) And _
     GetInput(eLinks) And _
     GetInput(eRechts) Then
    TeilOK = True
    For i = 1 To EA
      If GetInput(ePhoto) Then
        SetMotor mStanze, ftiEin
        WaitForHigh eEnde
        SetMotor mStanze, ftiAus
      Else
        SetMotor mStanze, ftiAus
        PrintLog "PhotoMist : " & Now
        TeilOK = False
        Beep 1111, 100
        Beep 555,  100
        Beep 1111, 100
        Exit For
      End If
    Next
    If TeilOK Then Produktion = Produktion + 1
  End If
  PrintStatus "Anzahl produzierte Teile : " _
                                & Produktion
Loop Until Finish
```

Die interessantesten Punkte des Programms sind :

1. Beim Start des Programms wird die Maschine auf **Ausgangslage** gefahren (Lichtschranke an, Stanze oben, Produktion = 0)
2. Ein **Stanzvorgang** kann nur ausgelöst werden, wenn die Lichtschranke geschlossen ist. Die Auslösung erfolgt durch "**Zweihandeinrückung**" : eLinks und eRechts gleichzeitig. Das ergibt dann ein gewaltiges If bei dem die einzelnen Bedingungen durch And verknüpft sind : alle Bedingungen müssen aufeinmal wahr sein. Achtung es werden hier Fortsetzungszeilen verwendet (_) am Zeilenende. Vor dem (_) muß eine Leerstelle stehen.
3. Neu ist auch die **For ... Next Schleife**, die die Anzahl der in EA vorgegebenen Hübe überwacht.
4. Vor jedem Hub wird die Lichtschranke nocheinmal kontrolliert und ggf. kräftig gemeckert und gehupt und mit Exit For die For ... Next Schleife und damit der **Stanzvorgang abgebrochen.**
5. Der einzelne Hub wird mit **WaitForHigh** kontrolliert. WaitForHigh wartet dass eEnde erst auf False und dann auf True wechselt. Das ist erforderlich, da ein einfaches Warten auf True sofort zum Erfolg führen kann, da der Taster vom letzten Hub noch auf True steht.

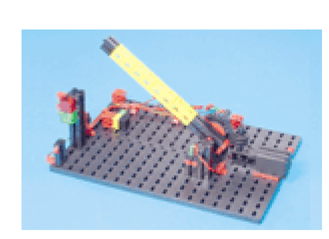
6. Nach jedem Stanzvorgang wird das **Produktionsergebnis** aufaddiert. Es werden aber nur die erfolgreichen Stanzvorgänge gezählt. Dazu die Variable TeilOK, die vor der For ... Next Schleife verdachtsweise auf True gesetzt wird und ggf. bei Abbruch im Zuge des Meckerns auf False geändert wird.
7. Die aktuelle **Produktion** wird in der Statuszeile und die **Zwischenfälle** werden im Log-Fenster angezeigt.

FishFace Referenz : WaitForHigh, Beep
VBScript Index : For..Next-Anweisung, Durchlaufen von Codeschleifen

---

Findige Bediener werden, der Sicherheitsmaßnahmen zum Trotz, doch noch ihre Finger in die laufende Maschine stecken können (die Abschaltung erfolgt erst nach einem Hub, es wird nicht geprüft, ob die Zweihandeinrückung vor jedem Stanzvorgang betätigt wird – ein "Dauerlauf" durch "festgeklemmte" Taster ist möglich). Hier soll es aber erstmal genügen.

Parkhausschranke



Motor an M1
Rote Lampe an M2
Grüne Lampe an M3
Lichtschranke an M4
ZuTaster an E1
AufTaster an E2
BedienTaster an E3
Photowiderstand E4

"Durch Betätigen des Tasters E3 soll die Schranke geöffnet werden. Ist die Schranke offen, leuchtet die Ampel grün. Erst wenn die Lichtschranke passiert wurde, springt die Ampel auf Rot und die Schranke schließt wieder"

```
' --- Parkhaus1.VBS ---
Option Explicit
Const mSchranke = 1, mRot = 2, mGruen = 3,
mLicht = 4
Const eZu = 1,       eAuf = 2, eOeffnen = 3,
ePhoto = 4
Const sZu = 1,       sAuf = 2

SetMotor mLicht,    ftiEin
SetMotor mRot,      ftiEin
Do
  PrintStatus  "--- Schranke schließt ---"
  SetMotor     mSchranke, sZu
  WaitForInput            eZu
  SetMotor     mSchranke, ftiAus
  PrintStatus  "--- Wartet auf Kunden ---"
  WaitForHigh  eOeffnen
  PrintStatus  "--- Schranke öffnet ---"
  SetMotor     mSchranke, sAuf
  WaitForInput eAuf
  SetMotor     mSchranke, ftiAus
  SetMotor     mRot,      ftiAus
  SetMotor     mGruen,    ftiEin
  PrintStatus  "--- Warten auf Durchfahrt ---"
  WaitForLow   ePhoto
  WaitForHigh  ePhoto
  SetMotor     mGruen,    ftiAus
  SetMotor     mRot,      ftiEin
  WaitForTime  500
Loop Until Finish
```

Das zugehörende Programm ist eine schöne Sammlung bekannter Elemente, lediglich ein **WaitForLow** als Pendant zu WaitForHigh (diesmal : True/False Durchgang) hat sich eingefunden. Mit der Kombination WaitForLow/WaitForHigh wird kontrolliert ob ein Auto in die Lichtschranke ein- und wieder ausgefahren ist. Sicherheitshalber wird danach noch ein wenig gewartet. Die eingestreuten PrintStatus erklären den weiteren Ablauf.

Spannend wird das, wenn anstelle eines schlichten Taster-Drucks ein PIN eingegeben werden muß :

```
  PrintStatus  "--- Wartet auf Kunden ---"
  If WaitForCode Then
    PrintStatus  "--- Schranke öffnet ---"
```

```
    .....
    WaitForTime  500
  End If
Loop Until Finish

Function WaitForCode
  WaitForCode = True
  If Secret(InputBox("Bitte Zugangscode eingeben!", _
                     "ftParkhaus GBR")) Then Exit Function
  If MsgBox("Sorry - das wars nicht", _
            vbOKCancel+vbCritical, _
            "ftParkhaus GBR") = vbCancel Then
    NotHalt = True
    PrintLog "Das Programm wurde gewaltsam beendet " & Now
  End If
  WaitForCode = False
End Function

Function Secret(PINcode)
  Secret = False
  If PINcode = CStr(Day(Now)*100 + Month(Now)) Then Secret = True
End Function
```

Anstelle des bisherigen WaitForHigh ein If WaitForCode Then das als Ergebnis ein True oder False zurückgibt. Im True-Falle geht's weiter wie bisher, sonst wird der Rest übersprungen und man landet wieder beim WaitForCode. Die Funktion WaitForCode sieht zwar so aus wie die anderen WaitFor-Funktionen (damit man sie auch ernst nimmt), ist aber eine Funktion, die gleich unten im Programm zu finden ist.

In WaitForCode gibt es als erstes eine Abfrage nach der PIN. Das geschieht über die (System)Funktion **InputBox**, zurückgegeben wird der eingegebene PIN-Code. Die Parameter von InputBox bedeuten der Reihe nach : Eingabeaufforderung, Text in der Titelzeile. Das Ergebnis wird einer hier erstellten Function **Secret** übergeben, diese wiederum vergleicht es mit dem intern gebildeten aktuellen PIN.

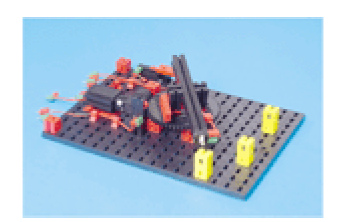
War der eingegebene PIN-Code falsch wird mit **MsgBox** gemeckert. Die Parameter : der Mecker-Spruch, die Button mit denen geantwortet werden kann + ein Icon mit dem die Meldung verziert wird und dann noch der Text in der Titelzeile.

Bei der Anwort Abbrechen (Cancel) wird eine "**Notbremse**" aktiviert die FishFace-Eigenschaft NotHalt wird auf True gesetzt, das wird dann auch noch protokolliert. Die Funktion wird wie bei Antwort OK mit False verlassen. In beiden Fällen wird dann der nachfolgende Code übersprungen. Aber im Fall NotHalt wird am Loop-Ende Finish hellhörig und bricht das Programm ab.

Und dann wäre da noch die Function **Secret**, die die aktuelle PIN liefert und mit dem als Parameter übergebenen PIN-Code aus der InputBox vergleicht, ja die wird aus Tag und Monat zusammengebastelt (heute war es 1303 und das war im März). Man kann die Sache auch noch viel spannender machen.

FishFace Referenz : WaitForLow
VBScript Index : InputBox-Funktion, MsgBox-Funktion

Der Schweißroboter



Motor an M1
Lampe an M2
EndeTaster an E1
ImpulsTaster an E2

"Der Roboter soll drei Positionen anfahren und an jeder Position eine Schweißung durchführen. Danach soll er in seine Ausgangsposition zurückkehren und von vorne beginnen."

```
' --- SchweissRobot1.VBS ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Const eEnde = 1, eImpuls = 2
Dim IstPosition

Do
  SetMotor       mRobot, ftiLinks
  WaitForInput   eEnde
  SetMotor       mRobot, ftiAus
  IstPosition    = 0

  SetMotor       mRobot,  ftiRechts
  WaitForPosUp   eImpuls, IstPosition, 56
  SetMotor       mRobot,  ftiAus
  Schweissen

  SetMotor       mRobot,  ftiRechts
  WaitForPosUp   eImpuls, IstPosition, 144
  SetMotor       mRobot,  ftiAus
  Schweissen

  SetMotor       mRobot,  ftiLinks
  WaitForPosDown eImpuls, IstPosition, 94
  SetMotor       mRobot,  ftiAus
  Schweissen
Loop Until Finish

Sub Schweissen
Dim i
  For i = 1 To EA
    SetMotor mSchweiss, ftiEin
    Pause 100 * EB
    SetMotor mSchweiss, ftiAus
    Pause 100 * EB
  Next
End Sub
```

Es geht alles schön der Reihe nach. Begonnen wird mit dem Anfahren der Ausgangsposition (Home Position) um einen Bezugspunkt zu gewinnen. Das ist IstPosition = 0.

Anschließend folgen drei sehr ähnliche Anweisungsblöcke mit denen die gewünschten Positionen angefahren werden. Motor einschalten (Richtung beachten), Mit dem neuen Befehl **WaitForPosUp** / **WaitForPosDown** darauf warten, daß die als Konstante vorgegebene Zielposition erreicht wird (d.h. Zielposition = IstPosition wird), Motor wieder abschalten und in Ruhe "Schweissen" (Unterprogramm Schweissen).

WaitForPosUp zählt die IstPosition solange hoch, bis die Zielposition erreicht ist, WaitForPosDown zählt sie herunter. Gezählt werden Impulse am Taster eImpuls. Um den Motor kümmert sich der Befehl nicht.

FishFace Referenz : WaitForPosUp, WaitForPosDown

Das geht auch anders

Relatives Positionieren – Asynchrones Fahren

```
' --- SchweissRobot2.VBS ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Const eEnde = 1, eImpuls = 2

Do
  SetMotor      mRobot, ftiLinks
  WaitForInput  eEnde
  SetMotor      mRobot, ftiAus

  SetMotor      mRobot, ftiRechts, ftiFull, 56
  WaitForMotors 0,      mRobot
  Schweissen

  SetMotor      mRobot, ftiRechts, ftiFull, 88
  WaitForMotors 0,      mRobot
  Schweissen

  SetMotor      mRobot, ftiLinks, ftiFull, 50
  WaitForMotors 0,      mRobot
  Schweissen
Loop Until Finish
```

Der Programmaufbau ist der gleiche, aber der ach so bekannte SetMotor hat noch mehr Parameter als man so denkt. SetMotor mRobot, ftiRechts, ftiFull, 56. Die ersten Parameter sind bekannt, ftiFull als Geschwindigkeitskeitsangabe ist auch schon mal vorgekommen. Neu ist der letzte Parameter, der gibt die Anzahl Impulse an, die der Motor in die vorgegebene Richtung fahren soll. Er tut das asynchron, d.h. ohne, daß das Programm anhält. Deswegen das nachfolgende **WaitForMotors** mit dem auf das Erreichen der vorgegebenen Position gewartet wird, ein Abschalten ist aber nicht erforderlich.

SetMotors setzt mit dieser Parameterliste einen RobotMotor voraus. D.h. einen Motor mit festzugeordneten Tastern. Bei M1 sind das E1 als Endtaster und E2 als Impulstaster. Der Endtaster muß außerdem mit ftiLinks angefahren werden können. Für die weiteren Motoren gilt entsprechend M2 : E3/E4, M3 : E5/6 ... WaitForMotors kann auch auf mehrere Motoren gleichzeitig warten :

```
SetMotor ftiM1, ftiRechts, ftiFull, 123
SetMotor ftiM4, ftiLinks,  ftiHalf, 34
WaitForMotors 0, ftiM1, ftiM4
```

Hier wird darauf gewartet, daß M1 123 Impuls mit ftiFull nach rechts und M4 34 Impulse mit ftiHalf nach links macht.

Die Positionsangaben sind **relativ** d.h. sie beziehen sich auf die aktuelle Postion. Zunächst werden, von IstPosition = 0 ausgehend, 56 Impulse nach rechts gefahren, ZielPosition ist 56, dann 88 Impulse, ZielPosition ist dann 56+88 = 144. Die nächsten 50 Impulse gehen nach links, das ergibt eine ZielPosition von 144-50 = 94.

FishFace Referenz : RobMotoren, SetMotor, WaitForMotors

Es geht aber noch einfacher, auch wenns erstmal komplizierter ist :

Absolute Positionierung

Die hatten wir ja eigentlich schon in der ersten Version hier wird es viel komplizierter, aber technisch anspruchsvoller gelöst. Verbunden mit einer Verlagerung in ein Unterprogramm wird die "Nutzanwendung" erstaunlich kurz und übersichtlich :

```
' --- SchweissRobot3.VBS ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Dim   RobotPos

Do
  Home
  MoveTo     56
  Schweissen EA, EB
  MoveTo     144
  Schweissen EA/2, EB*2
  MoveTo     94
  Schweissen EA, EB
Loop Until Finish

Sub MoveTo (ZielPos)
  If RobotPos < ZielPos Then
    SetMotor mRobot, ftiRechts, ftiFull, ZielPos - RobotPos
    Do
      PrintStatus "RobotPos : " & ZielPos - GetCounter(eImpuls)
    Loop While WaitForMotors(100, mRobot) = ftiTime
    RobotPos = ZielPos + GetCounter(eImpuls)
  Else
    SetMotor mRobot, ftiLinks, ftiFull, RobotPos - ZielPos
    Do
      PrintStatus "RobotPos : " & ZielPos + GetCounter(eImpuls)
    Loop While WaitForMotors(100, mRobot) = ftiTime
    RobotPos = ZielPos + GetCounter(eImpuls)
  End If
  PrintStatus "RobotPos : " & RobotPos
End Sub

Sub Schweissen (Mal, Dauer)
Dim i
  For i = 1 To Mal
    SetMotor mSchweiss, ftiEin
    Pause 100 * Dauer
    SetMotor mSchweiss, ftiAus
    Pause 100 * Dauer
  Next
End Sub

Sub Home
  PrintStatus "--- Es geht nach Hause ---"
  SetMotor mRobot, ftiLinks
  WaitForInput eEnde
  SetMotor mRobot, ftiAus
  RobotPos = 0
  PrintStatus "--- Zu Hause ---"
End Sub
```

Das Unterprogramm **MoveTo** unterscheidet zunächst, ob es nach links oder rechts geht. Anhand dessen wird die Anzahl Impulse bestimmt die mit SetMotor zu fahren sind und dann, wie gewohnt, mit WaitForMotors auf die Fertigmeldung gewartet. Hier wird WaitForMotors aber gesagt nicht gleich auf fertig zu warten sondern nur 100 MilliSekunden (die 0 von vorher bedeutet endlos warten) und dann mit einem Returncode zurückzukehren. Abgefragt

wird ftiTime : Ablauf der vorgegebenen Zeit. Es gibt auch noch ftiEnde, ftiNotHalt und ftiEsc. Das geschieht in einer Schleife, bis die Bedingung ftiTime nicht mehr zutrifft. Hier nimmt man schlicht an, daß alles gut gegangen ist (ftiEnde). Hier nocheinmal der Hinweis, es kann auch auf bis zu 8 Motoren gleichzeitig gewartet werden. Das Programm wird dann allerdings ein wenig komplizierter (siehe FishFa30 Handbuch für Visual Basic 6).

Da hier in einer Schleife gewartet wird, kann da auch etwas getan werden. Hier z.B. die Anzeige der aktuellen Position in der Statuszeile. Der Ordnung halber wird sie zum Schluß noch einmal auf "Vordermann gebracht", es wird ja nur alle 100 Millisekunden abgefragt. Helfen tut dabei die Funktion **GetCounter,** die den aktuellen Stand der Impulszählung zurückgibt (immer positiv, gerechnet vom Startwert heruntergezählt auf 0).

Das Unterprogramm Schweißen wurde etwas modernisiert, man die einzelnen Punkte jetzt mit unterschiedlichen Werten schweißen. Die Befehle zum Anfahren der Ausgangsposition wurden in ein Unterprogramm verlagert.

FishFace Referenz : GetCounter

Jetzt könnte man natürlich auf die Idee kommen, den Schweißroboter durch Austausch der Lampe durch einen Photowiderstand in einen Lichtsuchroboter umzuwandeln und dann die gemessenen Lichtwerte samt Position in der Warteschleife auszugeben ....

Referenz

Allgemeines

Verwendete Parameterbezeichnungen

In der Referenz werden für Parameter und Returnwerte besondere Bezeichnungen verwendet um deren Bedeutung zu charakterisieren. Sie stehen gleichzeitig für einen Variablentyp bzw. alternativ eine Enum.

AnalogNr	Nummer eines Analog-Einganges (0-1, ftiNr)
AnalogWert	Rückgabewert beim Auslesen von EX/EY (0-1024)
Counter	Wert eines ImpulsCounters
Direction	Schaltzustand eines M-Ausganges (0-2, ftiDir)
InputNr	Nummer eines E-Einganges (0-8(16), ftiNr)
InputStatus	Rückgabewert beim Auslesen aller E-Eingänge (0 - &HFFFF)
LampNr	Nummer eines "halben"-M-Ausganges (0-8(16), ftiNr)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1 (0000 normal, 0001 RobMode).
MotorNr	Nummer eines M-Ausganges (0-4(8), ftiNr)
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts)
mSek	Zeitangabe in MilliSekunden
NrOfchanges	Anzahl Impulse
OnOff	Ein/Ausschalten eines M-Ausganges (Boolean, ftiDir)
PortName	Name des wählbaren Anschlußports (String) ("LPT", "COM1" – "COM8", "LPT1" – "LPT3")
Position	Position in Impulsen ab Endtaster
Speed	Geschwindigkeitsstufe mit der ein M-Ausgang (Motor) betrieben werden soll (0-15, ftiSpeed)
SpeedStatus	Status der Geschwindigkeiten aller M-Ausgänge Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1 Werte 0000 (stop) – 1111 (full)
TermInputNr	Nummer eines E-Einganges mit der die (Wait)Methode beendet werden soll (ftiNr)
Value	Allgemeiner numerischer Wert
WaitWert	Rückgabewert von WaitForMotors (ftiWait)

Die Aufrufparameter sind Werteparameter, d.h. sie übergeben Werte, geben aber keine zurück, Ausname Parameter IstPosition bei WaitForPosDown / WaitForPosUp

Eine Reihe von Parametern sind optional, d.h. sie brauchen nicht angegeben werden. Intern werden sie dann mit sinnvollen Werten belegt. Das wird beim einzelnen Befehl beschrieben.

Symbolische Konstanten

Um ein Programm lesbarer zu machen, werden von mscFish eine Reihe von symbolischen Konstanten angeboten, die in der Datei Global.VBS zusammengefaßt sind, sie können bei Bedarf geändert werden. Die Konstanten werden unter den folgenden Oberbegriffen (die nicht selber verwendet werden können) zusammengefaßt :

ftiDir	Angabe des Schaltzustandes (Drehrichtung, Ein/Aus) ftiLeft, ftiRight, ftiOff, ftiOn, ftiLinks, ftiRechts, ftiAus, ftiEin
ftiNr	Angabe der Nummer eines Ein- bzw. Ausganges ftiEX, ftiEY, ftiE1 .. ftiE8, ftiM1 .. ftiM4
ftiSpeed	Angabe einer Geschwindigkeitsstufe ftiNone = 0, ftiHalf = 7, ftiFull = 15, Zwischenwerte können numerisch eingegeben werden.
ftiWait	Returnwerte der Methode WaitForMotors ftiEnde, ftiEnd, ftiTime, ftiNotHalt, ftiEsc

Befehle

Allgemeines

Abbrechbar

Länger laufende Methoden (Wait...) können von außen durch Setzen der Eigenschaft NotHalt = True oder durch Drücken der Esc-Taste abgebrochen werden.

Wird bei den betroffenen Methoden besonders angegeben.

NotHalt

Die Eigenschaft NotHalt (siehe auch "Abbrechbar") wird intern genutzt um langlaufende Funktionen ggf. Abzubrechen. NotHalt wird von OpenInterface auf False gesetzt. Es kann im Programm (z.B. über einen HALT-Button) genutzt werden um den Programmlauf abzubrechen oder auch eine Endlosschleife (z.B. Do ... Loop Until ft.Finish) zu beenden. Soll das Programm anschließend weiterlaufen, so ist NotHalt wieder auf False zu setzen.

Speed

Die Geschwindigkeitssteuerung beruht auf einem zyklischen Ein- und Ausschalten der betroffenen M-Ausgänge (Motoren) im Takt des PollIntervals. Dazu wird intern für jede Geschwindigkeitsstufe eine entsprechende Schaltliste vorgehalten. Die Geschwindigkeitsstufe wird durch die Parameter Speed bzw. SpeedStatus für einen bzw. alle Motoren bei der Methode SetMotor(s) bestimmt.

Counter

Zu jedem E-Eingang wird ein Counter geführt, in dem die Impulse (Umschalten von True auf False und umgekehrt) gezählt werden. Die Counter werden bei OpenInterface auf 0 gesetzt. Sie werden außerdem von einigen Methoden interngenutzt (SetMotor, WaitForxxx). Sie können mit GetCounter abgefragt und mit SetCounter / ClearCounter(s) gesetzt werden. In der Regel werden sie zur Positionsbestimmung eingesetzt.

RobMotoren

Unter RobMotoren wird eine Kombination von einem M-Ausgang und zwei E-Eingängen mit den Funktionen Endtaster / Impulstaster verstanden, die im Betrieb eine Einheit bilden. Dabei muß am M-Ausgang ein Motor angeschlossen sein und an den E-Eingängen Taster mit Schließfunktion (Kontakte 1-3). Auf der Motorwelle muß ein "Impulsrädchen" montiert sein, das den Impulstaster betätigt. Der Motor muß linksdrehend angeschlossen werden. D.h. er läuft bei ftiLinks auf den Endtaster zu. Folgende Kombinationen sind zulässig

Motor	Endtaster	Impulstaster
1	1	2
2	3	4
3	5	6
4	7	8

Die RobMotoren können über die Methode

```
SetMotor MotorNr, Direction, Speed, Counter
```

betrieben werden. Die Methode erlaubt das simultane Anfahren vorgegebener Positionen mit bis zu 8 Motoren. Bei Erreichen einer Position wird der zugehörende Motor abgeschaltet. Die Methode ist asynchron. D.h.Das Erreichen der vorgegebenen Positionen wird von der Methode nicht abgewartet (der Ausführungsteil der Methode läuft in einem separatem Thread). Die Synchronität zum Programm kann durch die Methode WaitForMotors wieder hergestellt werden.

Beispiel

```
SetMotor ftiM1, ftiLeft, ftiHalf, 50
SetMotor ftiM2, ftiRight, ftiFull, 60
WaitForMotors 0, ftiM1, ftiM2
```

Motoren M1 und M2 werden gestartet, anschließend wird auf das Erreichen der Positionen gewartet.

Lampen am Interface

Die vier M-Ausgänge des Interfaces sind primär zum Schalten von Motoren in zwei Laufrichtungen vorgesehen. Doch bietet sich zusätzlch die Möglichkeit, Geräte, die nur ein- und ausgeschaltet werden müssen, an nur einem Pol eines M-Ausganges und Masse anzuschließen. Auf diese Weise ist es möglich z.B. acht Lampen mit einem Interface zu schalten. Hiefür gibt es die Methode SetLamp.

Sollen Lampen gemeinsam (z.B. bei einer Verkehrampel) geschaltet werden, können sie auch über die Methode SetMotors geschaltet werden. Dazu sind die einzelnen Lampenbits im MotorStatus zu setzen.

Es gibt Unterschiede zwischen den Interfaces :

Universal (paralleles) Interface : im nicht geschalteten Zustand sind die Lampen aus. M1 vorderer (gelber) Kontakt : Lampe 1, hinterer (oranger) Kontakt Lampe 2 ...

Intelligent (serielles) Interface : im nicht geschalteten Zustand sind die Lampen **an**. M1 vorderer Kontakt : Lampe 1, hinterer Kontakt Lampe 2 ...

Liste der Befehle

Beep

Ausgabe eines Tones auf dem Systemlautsprecher

Beep Frequenz, Dauer

Frequenz : Tonhähe in Hz
Dauer : Dauer der Tones in Millisekunden

Beispiel
```
Beep 432, 1000
```

Der Kammerton a quäkt 1 Sekunde lang auf dem Systemlautsprecher.

ClearCounter

Löschen des angegebenen Counters (0)

ClearCounter InputNr

Siehe auch ClearCounters, GetCounter, SetCounter

ClearCounters

Löschen aller Counter (0)

ClearCounters

Siehe auch ClearCounter, GetCounter, SetCounter

ClearLog

Das Log-Fenster der IDE wird gelöscht

ClearLog

Siehe PrintLog, PrintStatus

ClearMotors

Abschalten aller M-Ausgänge (ftiAus)

ClearMotors

Siehe auch SetMotor, SetMotors, SetLamp, Outputs

CloseInterface

Schließen der Verbindung zum Interface
-> Im Betrieb mit mscFish nicht erforderlich, aber beim Einsatz von VBScript-Programmen außerhalb der Umgebung von mscFish.

ft.**CloseInterface**

Siehe auch OpenInterface

EA

Auslesen des Wertes im Feld EA der IDE

wert = **EA**

Beispiel
```
AnzahlRunden = EA + 1
```

Die Variable Anzahlrunden wird mit dem numerischen Inhalt des IDE Feldes EA + 1 besetzt.

Siehe auch EB

EB

Auslesen des Wertes im Feld EB der IDE

wert = **EB**

Siehe auch EA

Finish

Feststellen eines Endewunsches (NotHalt, Esc-Taste, E-Eingang)

Boolean = **Finish**(InputNr)

Siehe auch GetInput, GetInputs, Inputs

Beispiel
```
Do
   ....
Loop Until Finish(ftiE1)
```

Die Do Loop-Schleife wird solange durchlaufen, bis entweder NotHalt = True, die Esc-Taste gedrückt oder E1 = True wurde.

GetAnalog

Feststellen eines Analogwertes (EX / EY).
Es wird der intern vorliegende Wert ausgegeben. AnalogScan beim OpenInterface ist erforderlich.

AnalogWert = **GetAnalog**(AnalogNr)

Siehe auch GetAnalogs, AnalogsEX, AnalogsEY, AnalogScan, OpenInterface

Beispiel
```
PrintStatus ft.Analog(ftiEX)
```

Der aktuelle EX-Wert wird in der Statuszeile angezeigt.

GetCounter

Feststellen des Wertes des angegebenen Counters

Counter = **GetCounter**(InputNr)

Siehe auch SetCounter, ClearCounter, ClearCounters

Beispiel

Beipiel
```
PrintStatus "Turm Position : " & GetCounter(ftiE2)
```

Die aktuelle Turm Position wird in der Statuszeile angezeigt.

GetInput

Feststellen des Wertes des angegebenen E-Einganges

Boolean = **GetInput**(InputNr)

Siehe auch GetInputs, Inputs, Finish

Beispiel
```
If GetInput(ftiE1) Then
   ...
Else
   ...
EndIf
```

Wenn der E-Eingang E1 (Taster, PhotoTransistor, Reedkontakt ...) = True ist, wird der Then-Zweig durchlaufen.

GetInputs

Feststellen der Werte aller E-Eingänge

InputStatus = **GetInputs**()

Siehe auch GetInput, Inputs, Finish

Beispiel
```
Dim e
   e = GetInputs
   If (e And &H1) Or (e And &H4) Then ...
```

Wenn die E-Eingänge E1 oder E3 True sind, wird der Then-Zweig ausgeführt.

NotHalt

Anmelden eines Abbruchwunsches, Auswertung durch die Wait-Methoden und Finish

Get | Set, Boolean, Default = False

Beispiel
```
If NotHalt Then Exit Sub
```

Das Unterprgramm wird beendet, wenn NotHalt den Wert True hat

Siehe auch Finish

OpenInterface

Herstellen der Verbindung zum Interface und Setzen/Bestimmen von dazu erforderlichen Parametern. OpenInterface muß deswegen als erste Methode aufgerufen werden.
-> Im Betrieb mit mscFish nicht erforderlich, aber beim Einsatz von VBScript-Programmen außerhalb der Umgebung von mscFish.

ft.**OpenInterface**(PortName, AnalogScan, Slave, PollInterval, LPTAnalog, LPTDelay)

Die Parameter ab AnalogScan sind optional :

- PortName (String) : LPT | COM1 – COM8 | LPT1 – LPT3
  Zuordnung des zuverwendeden Interfaces durch Angabe des Ports an dem es angeschlossen ist.
- AnalogScan (Boolean) : Angabe, ob auch die Analogeingänge (EX / EY) gepollt werden sollen. Es ist dann ein höheres PollInterval erforderlich. Default : False
- Slave (Boolean) : Angabe ob an das primäre Interface ein weiteres angeschlossen ist. Default = False
- PollInterval (Long) : Angabe in mSek in welchen Intervallen auf das Interface zugegriffen werden soll. Default = 0 : Bestimmung durch die Methode OpenInterface in Abhängigkeit vom Kontext.

- LPTAnalog (Long) : AnalogSkalierung. Begrenzung des Analogwertes nach oben und damit der für die Messung erforderlichen Zeit (nur LPT-Interface). Default = 5
- LPTDelay (Long) : Ausgabeverzögerung. Bei LPT-Interface auf schnellen Rechnern erforderlich. Default = 10

Siehe auch CloseInterface

Pause

Anhalten des Programmablaufs.

Pause mSek

Siehe auch WaitForTime

Beispiel
```
SetMotor ftiM1, ftiLinks
Pause 1000
SetMotor ftiM1, ftiAus
```

Der Motor am M-Ausgang M1 wird für eine Sekunde (1000 MilliSekunden) eingeschaltet.

PrintLog

Ausgabe eines Textes in das Log-Feld der IDE

PrintLog stringausdruck

Beispiel
```
PrintStatus "Schranke geöffnet : " & Now
```

Im Log-Fenster wird ausgegeben : `Schranke geöffnet : 16.03.2003 16:30:34`

Siehe auch PrintStatus

PrintStatus

Ausgabe eines Textes in die Statuszeile der IDE

PrintStatus stringausdruck

Beispiel
```
PrintStatus "Temperatur : " & (1000 - GetAnalog(aNTC)) / 10 - 12
```

In die Statuszeile der IDE wird Temperatur : 37,4 ausgegeben. GetAnalog(aNTC) ergab den Wert 506.

Siehe auch PrintLog

SetCounter

Setzen eines Counters

SetCounter InputNr, Value

Siehe auch GetCounter, ClearCounter, ClearCounters

SetLamp

Setzen eines "halben" M-Ausganges. Anschluß einer Lampe oder eines Magneten ... an einen Kontakt eines M-Ausganges und Masse. Siehe auch "Lampen am Interface"

SetLamp LampNr, OnOff

Siehe auch SetMotor, SetMotors, ClearMotors

Beispiel

```
Const lGruen = 1, lGelb = 2, lRot = 3

   SetLamp lGruen, ftiEin
   Pause 2000
   SetLamp lGruen, ftiAus
  SetLamp lGelb, ftiEin
```

Die grüne Lampe an M1-vorn und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an M1-hinten ...

SetMotor

Setzen eines M-Ausganges (Motor)

SetMotor MotorNr, Direction, Speed, Counter

Die Parameter ab Speed sind optional

MotorNr (ftiNr) : Nummer des zu schaltenden M-Ausganges

Direction (ftiDir) : Schaltzustand (ftiLinks, ftiRechts, ftiEin, ftiAus)

Speed (ftiSpeed) : Geschwindigkeitsstufe, Default : ftiFull

Counter (ftiNr) : Begrenzung der Einschaltzeit. Default = 0, unbegrenzt. Werte > 0 geben die Anzahl Impulse an, die der M-Ausgang eingeschaltet sein soll (Fahren eines Motors um n Impulse). Siehe auch "RobMotoren"

Siehe auch SetMotors, ClearMotors, SetLamp, Outputs

Beispiel 1

```
SetMotor ftiM1, ftiRechts, ftiFull
Pause 1000
SetMotor ftiM1, ftiLinks, ftiHalf
Pause 1000
SetMotor ftiM1, ftiAus
```

Der Motor am M-Ausgang M1 wird für 1000 MilliSekunden linksdrehend, volle Geschwindigkeit eingeschaltet und anschließen für 1000 mSek rechtsdrehend, halbe Geschwindigkeit.

Beispiel 2

```
SetMotor ftiM1, ftiLeft, 12, 123
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am E-Eingang E2 oder E1 = True mit Geschwindigkeitstufe 12 eingeschaltet. Das Abschalten erfolgt selbsttätig.

SetMotors

Setzen des Status aller M-Ausgänge

SetMotors MotorStatus, SpeedStatus, ModeStatus

Die Parameter ab SpeedStatus sind optional

MotorStatus (Long) : Schaltzustand der M-Ausgänge

SpeedStatus (Long) : Geschwindigkeitsstufen der M-Ausgänge. Default : ftiFull

ModeStatus (Long) : Betriebsmodus der M-Ausgänge. Default = 0, normal
Bei ModeStatus RobMode sind vorher mit SetCounter die zugehörenden Counterstände zu setzen.

Siehe auch ClearMotors, SetMotor, SetLamp, Outputs

Beispiel

```
SetMotors &H1 + &H80
Pause 1000
ClearMotors
```

Der M-Ausgang (Motor) M1 wird auf links geschaltet und gleichzeitig M4 auf rechts. Alle anderen Ausgänge werden ausgeschaltet. Nach 1 Sekunde werden alle M-Ausgänge abgeschaltet.

WaitForChange

Warten auf eine vorgegebene Anzahl von Impulsen

WaitForChange InputNr, NrOfChanges, TermInputNr

Der Parameter TermInputNr ist optional

InputNr (ftiNr) : E-Eingang an dem die Impulse gezählt werden.

NrOfChanges (Long) : Anzahl Impulse

TermInputNr (ftiNr) : Alternatives Ende. E-Eingang = True

Intern wird Counter (InputNr) verwendet, der zu Beginn der Methode zurückgesetzt wird

Siehe auch WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiLeft
WaitForChange ftiE2, 123, ftiE1
SetMotor ftiM1, ftiOff
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an E-Eingang E2 oder E1 = True gewartet, der Motor wird abgeschaltet. Vergleiche mit dem Beispiel bei SetMotor. Hier wird das Programm angehalten solange der Motor läuft.

WaitForHigh

Warten auf einen False/True-Durchgang an einem E-Eingäng

WaitForHigh InputNr

Siehe auch WaitForLow, WaitForChange, WaitForInput

Beispiel

```
SetMotor ftiM1, ftiOn
SetMotor ftiM2, ftiLeft
WaitForHigh ftiE1
SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband aus der Lichtschranke ausgefahren ist (die Lichtschranke wird geschlossen), dann wird abgeschaltet. Die Lichtschranke muß vorher False sein (unterbrochen).

WaitForInput

Warten daß der angegebene E-Eingang den vorgegebenen Wert annimmt.

WaitForInput InputNr, OnOff

OnOff (Boolean) : Endebedingung für E-Eingang InputNr, Default = True

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiLeft
WaitForInput ftiE1
SetMotor ftiM1, ftiOff
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf E-Eingang = True gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer Endposition.

WaitForLow

Warten auf einen True/False-Durchgang an einem E-Eingang

WaitForLow InputNr

Siehe auch WaitForChange, WaitForInput, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiOn
SetMotor ftiM2, ftiLeft
WaitForLow ftiE1
SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband in die Lichtschranke einfährt (sie unterbricht), dann wird abgeschaltet. Die Lichtschranke muß vorher True sein (nicht unterbrochen).

WaitForMotors

Warten auf ein MotorReadyEreignis oder den Ablauf von Time

WaitWert = **WaitForMotors**(Time, MotorNr .....)

Time (Long) : Zeit in MilliSekunden. Bei Time = 0 wird endlosgewartet.

MotorNr (ftiNr) : Nummern der M-Ausgänge auf die gewartet werden soll. Es wird auf MotorStatus = ftiAus für die angegebenen M-Ausgänge gewartet. MotorNr ftiM1 – ftiM8 in beliebiger Reihenfolge. Die nicht betroffenen Motoren müssen nicht angegeben werden. Bei den Return-Werten ftiWait.ftiNotHalt und .ftiESC werden alle betroffen Motoren angehalten.

Beispiel

```
SetMotor ftiM4, ftiLeft,  ftiHalf, 50
SetMotor ftiM3, ftiRight, ftiFull, 40
Do
  PrintStatus GetCounter(ftiE6) & " - " & GetCounter(ftiE8)
Loop While WaitForMotors(100, ftiM4, ftiM3) = ftiTime
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehend mit voller Geschwindigkeit für 40 Impulse. Die Do Loop-Schleife wartet auf das Ende der Motoren (WaitForMotors). Alle 100 MilliSekunden wird in der Schleife die aktuelle Position angezeigt (100 .... = ftiTime). Wenn die Position erreicht ist <> ftiTime, ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet. Achtung hier wurde nicht auf NotHalt (ftiNotHalt) oder Esc-Taste (ftiEsc) abgefragt, es könnte also auch vor Erreichen der Zielposition abgebrochen worden sein.

WaitForPosDown

Warten auf Erreichen einer vorgegebenen Position.

WaitForPosDown InputNr, IstPosition, ZielPosition, TermInputNr

Ausgegangen wird von der aktuellen Position, die in IstPosition gespeichert ist, es werden solange Impulse von IstPosition abgezogen, bis der in ZielPosition angegebene Stand erreicht ist. IstPosition enthält dann tatsächlich erreichte Position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = True beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosUp, WaitForChange

Beispiel

```
Dim IstPosition
  IstPosition     = 12
   SetMotor       ftiM1, ftiLinks
```

```
  WaitForPosDown ftiE2, IstPosition, 0
  SetMotor       ftiM1, ftiAus
```

Die aktuelle Position ist 12 (IstPosition), der Motor an M-Ausgang M1 wird linksdrehend gestartet. WaitForPosDown wartet dann auf Erreichen der Position 0, der Motor wird dann ausgeschaltet

WaitForPosUp

Warten auf Erreichen einer vorgegebenen Position.

WaitForPosUp InputNr, IstPosition, ZielPosition, TermInputNr

Ausgegangen wird von der aktuellen Position in IstPosition, es werden solange Impulse auf IstPosition addiert, bis der in ZielPosition angegebene Stand erreicht ist. IstPosition enthält dann tatsächlich erreichte Position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = True beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosDown, WaitForChange

Beispiel

```
Dim IstPosition
  IstPostion  = 0
  SetMotor     ftiM1, ftiRechts
  WaitForPosUp ftiE2, IstPosition, 24
   SetMotor     ftiM1, ftiAus
```

Die aktuelle Position ist 0 (IstPosition), der Motor an M-Ausgang M1 wird rechtsdrehend gestartet. WaitForPosUp wartet dann auf Erreichen der Position 24, der Motor wird dann ausgeschaltet. Siehe auch Beispiel zu WaitForPosDown, hier wird in Gegenrichtung gefahren.

WaitForTime

Anhalten des Programmablaufs.

WaitForTime mSek

Synonym für Pause

Siehe auch Pause

Beispiel

```
Do
  SetMotors &H1
  WaitForTime 555
  SetMotors &H4
  WaitForTime 555
Loop Until Finish
```

In der Schleife Do Loop Until Finish wird erst M-Ausgang (Lampe) M1 eingeschaltet und alle anderen abgeschaltet (binär : 0001), dann gewartet, M2 (Lampe) eingeschaltet (Rest aus, binär : 0100) und gewartet. Ergebnis ein Wechselblinker.

Anhang

Übersicht mscFish

mscFish ist eine Entwicklungumgebung für VBScript Programme, die FishFa30.DLL nutzen. Sie unterstützt die Erstellung von Anwendungen zur Steuerung von fischertechnik Modellen, die über ein fischertechnik Interface an einen Windows PC angeschlosen sind.

Einsetzbar auf Windows 98/Me bzw. Windows NT/2000/XP

mscFish besteht aus folgenden Komponenten :

1. Der Entwicklungsumgebung : mscFish30.EXE
2. Dem Handbuch dazu : mscFish30.PDF
3. Der VBScript-Dokumentation : Script56.CHM

Von diesen Komponenten werden folgende weitere Komponenten genutzt :

1. Das Visual Basic 6 Laufzeitsystem : MSVBVM60.DLL und davon abhängige Komponenten
2. Die ActiveX-Komponente FishFa30.DLL mit umFish30.DLL und weiteren Komponenten.
3. Der Acrobat Reader.
4. Das WSH-System. Der Windows Scripting Host mit seinen Komponenten
5. Ein Hilfe-System für CHM-Dateien

Diese Komponenten können wie folgt bereitgestellt werden :

Zu 1 : Ist auf den moderneren Windows-System standardmäßig installiert

Zu 2 : in mscFish30Setup.EXE enthalten

Zu 3 : ist meistens installiert (Version 4 reicht), kann ggf. aus dem Internet heruntergeladen werden : www.adobe.de/products/acrobat/readstep.html

Zu 4 : ist meistens installiert, kann ggf. aus dem Internet heruntergeladen werden. Benötigt werden die Pakete scr56de.exe für Win 98/ME/NT bzw. scriptde.exe für Win 2000/XP www.microsoft.com/germany/scripting

und dort rechts oben "Windows Script 5.6 freigegeben" und dann das Paket scd56de.exe (Stand März 2003).

Zu 5 : ist auf neueren Windows-Systemen installiert.

Folgende Files werden als SharedFiles nach ...\System32 installiert :

CMDLGDE.DLL	MSScript.OCX
COMDLG32.DLL	SCRrnde.DLL
	SCRrun.DLL

Übergang zu anderen Sprachen der VB-Sprachfamilie

Ein Übergang zu anderen Sprachen lohnt :

- wenn man seine Programme mit einer eigenen Oberfläche ausstatten will,
- wenn man komplexere Programme (also über die oben angegebenen "paar" Seiten hinauskommt) erstellen will,
- Programme wünscht, die ohne die IDE von mscFish laufen
- oder man Wert auf eine komfortable Entwicklungsumgebung legt.
- und ist erforderlich, wenn man Programme erstellen will, die den Slave/Extension Module oder mehrere Interfaces nutzen.

Dabei ist generell zu beachten, daß in diesem Fall selber eine Instanz von FishFa30.DLL selber zu erstellen ist z.B mit Set ft = CreateObject(FishFa30.FishFace). Die Instanzvariable – hier ft – ist dann allen FishFace-Befehlen voranzustellen.

Die Befehle, die direkt auf die IDE zugreifen (PrintLog, PrintStatus, ClearLog, EA / EB)sind durch geeignete der jeweiligen Sprache zu ersetzen. Die Befehle WaitForPosUP/Down sind in WaitForPositionUP/Down zu ändern.

Zu beachten ist, das die Einarbeitung in die neue Entwicklungsumgebung – Ausnahme VBScript pur – einigen Aufwand verlangen wird. Die Sprache Visual Basic und damit die in ihr geschriebenen Programme, bleiben erhalten.

Beispiel für Programme in anderen Sprachen finden sich auf www.ftcomputing.de

VBScript pur

Ist ohne Installation neuer Software möglich, lohnt aber nur, wenn man Programme ohne eigene Oberfläche erstellen will, sie laufen dann "unsichtbar" im Hintergrund. Zu beachten ist, daß sich die Befehle PrintStatus und PrinLog schlecht ersetzen lassen.

VBScript und HTML

Wenn man Erfahrung mit der Erstellung von HTML-Seiten (und vielleicht auch Tools dazu hat), ist es interessant Programme mit eigener HTML-Oberfläche zuerstellen, die die vorhandenen VBScript-Routinen nutzen.

VBA – Visual Basic for Applications

Wenn auf dem Rechner bereits MS WinWord oder Excel installiert ist, hat man mit VBA bereits ein vollwertiges Entwicklungssystem, das auch die Entwicklung eigener Oberflächen gestattet. Lediglich der Zugang über WinWord und das Menü Extras | Makros ist etwas umständlich.

Visual Basic 6

VB6 bietet in der preisgünstigen Einsteiger Edition ein vollwertiges, professionelles Entwicklungssystem und läßt sich durch die deutlich teurere Professionelle Edition noch durch weitere Funktionalität ausstatten.

Visual Basic .NET

VB.NET ist eine Sprache der .NET Sprachfamilie, die auf Microsofts .NET Framework basiert. Ein Umstieg ist besonders dann reizvoll, wenn man die neue Technologie kennenlernen will.